Problémy a algoritmy v kostce

14. června 2006

1 Problémy

Kombinatorický problém je charakterizován

- vstupními proměnnými (na příkladu s vrtačkou a tišťákem je to seznam děr),
- výstupními proměnnými (pořadí děr tak, jak se budou vrtat),
- konfiguračními proměnnými (také pořadí děr),
- omezením (každou díru vrtat jen jednou, uzavřená cesta),
- optimalizačním kritériem, je-li třeba (co nejkratší cesta vrtačky).

Instance problému je ohodnocení vstupních proměnných.

Konfigurace je ohodnocení konfiguračních proměných.

Řešení instance je konfigurace, která splňuje omezení.

Optimální řešení je řešení s nejlepší hodnotou optimalizačního kritéria.

Suboptimální řešení je řešení s *vyhovující* hodnotou optimalizačního kritéria. Otázkou je, co je vyhovující hodnota.

1.1 Typy kombinatorických problémů a jejich optimalizační verze

1.1.1 Čistě kombinatorické problémy

Notace: I je instance problému, Y je konfigurace, $R(I, Y)$ je omezení ($R(I, Y)$ tedy říká, jestli je Y řešením).

Rozhodovací problém Existuje Y takové, že $R(I, Y)$?

Konstruktivní problém Sestrojit nějaké Y takové, že $R(I, Y)$.

Enumerační problém Sestrojit všechna Y taková, že $R(I, Y)$.

1.1.2 Optimalizační problémy

K předchozí notaci ještě přibývá optimalizační kritérium (cenová funkce) $C(Y)$.

Optimalizační rozhodovací problém Existuje Y takové, že $R(I, Y)$ a $C(Y)$ je lepší než nějaká konstanta Q?

Optimalizační konstruktivní problém Sestrojit nějaké Y takové, že $R(I, Y)$ a $C(Y)$ je nejlepší možné.

Optimalizační enumerační problém Sestrojit všechna Y taková, že $R(I, Y)$ a $C(Y)$ je nejlepší možné.

Optimalizační evaluační problém Zjistit nejlepší možné $C(Y)$ takové, že $R(I, Y)$.

1.2 Složitost problému, algoritmu

1.2.1 Asymptotická složitost algoritmu

Nechť $f, g : \mathrm{N} \to \mathrm{R}^+$

Asymptotická horní mez $f(n)$ je shora omezena $g(n)$, $f(n) = O(g(n))$, pokud existuje kladné c takové, že pro všechna $n > n_0$ platí: $f(n) \leq c \cdot g(n)$.

Asymptotická dolní mez $f(n) = \Omega(g(n)) \iff \exists c > 0, n_0 \in \mathrm{N} : \forall n > n_0 : f(n) \geq c \cdot g(n)$.

Asymptotický odhad $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$, tzn. $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.

1.2.2 Složitost problému

Složitost problému se určuje jako složitost algoritmu, který ho řeší. Máme-li algoritmus $f(n)$, který řeší problém Π, pak je jeho složitost $O(f(n))$.

1.3 Stavový prostor

Stav Nechť $X = \{x_1, x_2, \ldots, x_n\}$ jsou konfigurační proměnné problému Π. Nechť $Z = \{z_1, z_2, \ldots, z_m\}$ jsou vnitřní proměnné algoritmu A řešícího instanci I problému Π. Pak každé ohodnocení s proměnných $X \cup Z$ je *stav* algoritmu A řešícího I.

Stavový prostor Nechť $S = \{s_i\}$ je množina všech stavů algoritmu A řešícího I. Nechť $Q = \{q_j\}$ je množina operací $S \to S$ takových, že $q_j(s_i) \neq s_i$ pro všechna s_i, q_j. Pak dvojici (S, Q) nazveme *stavovým prostorem* algoritmu A řešícího I.

1.3.1 Graf stavového prostoru

Tah Nechť $s \in S$ je (jeden konkrétní) stav a $q \in Q$ operace. Pak aplikace $q(s)$ se nazývá *tah*.

Graf stavového prostoru Nechť (S, Q) je stavový prostor algoritmu řešícího instanci problému. Pak orientovaný graf $H = (S, E)$, kde hrana $(s_i, s_j) \in E$ odpovídá tahu $s_j = q(s_i)$ pro $q \in Q$ se nazývá *grafem stavového prostoru* algoritmu.

1.3.2 Strategie pohybu stavovým prostorem

Po stavovém prostoru se pohybujeme transformací aktuálního stavu pomocí dostupných operací na nějaký jiný stav. Tuto transformaci je třeba řídit.

Úplná strategie Navštívíme všechny stavy kromě těch, o kterých víme, že nedávají (optimální) řešení.

Systematická strategie Úplná strategie, při které navštívíme každý stav nejvýše jednou.

Vlastností systematických strategií je, že jsou v nejhorším případě rovny hrubé síle. To nastane například v případě neexistujícího řešení. Výhodou je, že existuje-li řešení, tyto metody jej naleznou. Navíc naleznou řešení optimální.

Na základě dosud dosažené hodnoty optimalizačního kritéria, splnění omezujících podmínek nebo znalosti optimalizačního kritéria a omezujících podmínek můžeme vyloučit (prořezat) určité oblasti stavového prostoru (větve hledání).

1.3.3 Prohledávací prostor

Rozšíříme-li stavový prostor tak, že každému prvku nepřísluší jedna konfigurace, ale množina konfigurací, hovoříme o *prohledávacím prostoru*.

2 Třídy složitosti problémů

Na složitost problémů lze nahlížet různými způsoby. Jako *únosné* lze označit takové problémy, jajichž stavový prostor je polynomiálně velký. *Neúnosné* jsou pak takové, jejichž stavový prostor je nadpolynomiální. U některých existuje únosně dlouhá cesta stavovým prostorem k řešení, pokud víme, kudy jít. U některých ani to ne.

Více formalismu do celé problematiky se snaží vnést systém tříd problémů. Modelem univerzálního výpočetního stroje je tu *Turingův stroj*.

2.1 Třídy rozhodovacích problémů

2.1.1 Řešení problému deterministickým Turingovým strojem

Program M pro deterministický Turingův stroj *řeší rozhodovací problém* Π, jestliže se výpočet zastaví po konečném počtu kroků pro každou instanci problému Π.

Program M pro deterministický Turingův stroj *řeší rozhodovací problém* Π *v čase* t, jestliže se výpočet zastaví po t krocích pro každou instanci problému Π.

2.1.2 Třída P

Rozhodovací problém patří do *třídy P* jestliže pro něj existuje program pro deterministický Turingův stroj, který jej řeší v čase $O(n^k)$, kde n je velikost instance a k je konečné číslo.

2.1.3 Třídy PSPACE a EXPTIME

Pro problém spadající do třídy *PSPACE* platí, že spotřebuje polynomiální množství pásky. Problém ze třídy *EXPTIME* je řešen v čase $O(2^{P(n)})$, kde $P(n)$ je polynom ve velikosti instance n.

2.1.4 Řešení problému nedeterministickým Turingovým strojem

Program M nedeterministického Turingova stroje *řeší rozhodovací problém* Π *v čase* t, jestliže se výpočet zastaví po t krocích pro každou instanci $I \in \Pi_{ANO}$ problému Π, kde Π_{ANO} je množina instancí problému Π takových, které mají výstup ANO.

Jestliže nedeterministický Turingův stroj řeší problém Π v čase $T(n)$, pak deterministický Turingův stroj řeší Π v čase $2^{O(T(n))}$.

2.1.5 Třída NP

Rozhodovací problém Π *patří do třídy NP*, jestliže pro něj existuje program pro nedeterministický Turingův stroj, který každou instanci $I \in \Pi_{ANO}$ problému Π řeší v čase $O(n^k)$, kde n je délka vstupních dat a k je konečné číslo.

Rozhodovací problém Π *patří do třídy NP*, jestliže pro každou instanci $I \in \Pi_{ANO}$ problému Π existuje konfigurace Y taková, že kontrola, zda Y je řešením, patří do P (tzn. že omezující podmínky lze vyhodnotit v polynomiálním čase). V této souvislosti nazýváme Y *certifikátem*.

2.1.6 Třída co-NP

Existují i problémy mimo NP. Například rozhodovací problém, zda je graf prost Hamiltonových kružnic, nebo zda je booleovská formule nesplnitelná. Problémem je, že při odpovědi „ano, je nesplnitelná“ nemáme žádný certifikát, kterým bychom to doložili. Na takovéto problémy lze pohlížet jako na „protějšky“ NP problémů – označují se *co-NP*.

2.2 Třídy optimalizačních problémů

2.2.1 Řešení optimalizačního problému Turingovým strojem

Program M pro deterministický Turingův stroj *řeší optimalizační problém* Π *v čase* t, jestliže se výpočet zastaví po t krocích pro každou instanci problému Π, která má řešení, a na pásce je zapsán výstup instance.

Program M pro deterministický Turingův stroj *počítá optimalizační kritérium problému* Π *v čase* t, jestliže pro každé řešení každé instance problému Π, zapsané na pásce jako vstupní data, se výpočet zastaví po t krocích a na pásce je zapsána hodnota optimalizačního kritéria.

2.2.2 Třída NPO

Optimalizační problém Π *patří do třídy NPO*, jestliže splňuje následující podmínky:

- velikost výstupu instance je omezena polynomem ve velikosti instance (tj. výstup lze zapsat v polynomiálním čase),
- problém, zda daná konfigurace je řešením, patří do P (tj. omezující podmínky lze vyhodnotit v polynomiálním čase),
- existuje program pro Turingův stroj, který vypočítá hodnotu optimalizačního kritéria pro všechna řešení všech instancí v polynomiálním čase (tj. optimalizační kritérium lze vyhodnotit v polynomiálním čase).

2.2.3 Třída PO

Optimalizační problém Π *patří do třídy PO*, jestliže splňuje následující podmínky:

- patří do NPO,
- existuje program pro Turingův stroj, který každou instanci vyřeší v polynomiálním čase.

Příkladem je třeba problém nejkratší cesty v grafu $G = (V, E)$. Velikost výstupu je $O(|E|)$, ověřit omezení trvá $O(|E| \log |E|)$, optimalizační kritérium spočteme v čase $O(|E|)$ a existuje polynomiální algoritmus (Dijkstra).

2.3 NP-úplné a NP-těžké problémy

X-těžký Problém Π je *X-těžký*, jestliže se efektivní řešení všech problémů ze třídy X dá zredukovat na efektivní řešení problému Π. Efektivním řešením je například řešení v polynomiálním čase nebo řešení s omezenou chybou. Zredukovat na řešení Π znamená vyřešit pomocí Π.

X-úplný Problém Π je X úplný, jestliže je X-těžký a sám patří do třídy X.

2.3.1 NPC problémy

Karpova redukce Rozhodovací problém Π_1 je Karp-redukovatelný na Π_2 (značí se $\Pi_1 \propto \Pi_2$), jestliže existuje polynomiální program pro deterministický Turingův stroj, který převede každou instanci I_1 problému Π_1 na instanci I_2 problému Π_2 tak, že výstup obou instancí je shodný.

Karpova redukce je *tranzitivní*, tedy platí, že $(\Pi_1 \propto \Pi_2) \wedge (\Pi_2 \propto \Pi_3) \Rightarrow \Pi_1 \propto \Pi_3$. Zároveň lze pomocí Karpovy redukce vytvářet *třídy polynomiální ekvivalence* ($(\Pi_1 \propto \Pi_2) \wedge (\Pi_2 \propto \Pi_1) \Rightarrow \Pi_1$ a Π_2 jsou polynomiálně ekvivalentní).

NP-úplný problém Problém Π je NP-úplný (NPC, NP-Complete), jestliže $\Pi \in \mathrm{NP}$ a pro všechny problémy $\Pi' \in \mathrm{NP}$ platí, že $\Pi' \propto \Pi$.

Důsledkem je, že máme-li problém $\Pi \in \mathrm{NP}$ a existuje-li třeba i jediný $\Pi' \in \mathrm{NPC}$ takový, že $\Pi' \propto \Pi$, pak z tranzitivity plyne, že $\Pi \in \mathrm{NPC}$ (česky: když najdeme jeden NPC problém, který jde převést na náš problém Π, pak musí být náš problém taky NPC, jelikož všechny NP jdou převést na náš problém přes Π' ve dvou krocích).

NPC jsou nejtěžší, vzájemně převoditelné, problémy v NP.

Cookova věta SAT je NP-úplný.

Velice důležitá věta, která přináší řadu důsledků. Především říká, že NPC není prázdná a otevírá tak cestu k důkazům NP-úplnosti převodem. Jsou známy tisíce NPC problémů, které tvoří třídu ekvivalence. Nezdá se proto, že by třída problémů P byla shodná s NP.

Toho, že SAT je NP-úplný, a předchozího důsledku se využívá k dokazování, že je nějaký problém NP-úplný ($\Pi \in \mathrm{NP}, \mathrm{SAT} \propto \Pi \Rightarrow \Pi \in \mathrm{NPC}$).

2.3.2 NPH problémy

Turingova redukce Rozhodovací problém Π_1 je Turing-redukovatelný na Π_2 (značíme třeba $\Pi_1 \overset{\mathrm{T}}{\propto} \Pi_2$, jestliže existuje polynomiální program pro (deterministický) Turingův stroj, který řeší každou instanci I_1 problému Π_1 tak, že používá program M_2 pro problém Π_2 jako podprogram (jehož trvání považujeme za jeden krok).

Karpova redukce je speciálním případem Turingovy redukce, kdy voláme podprogram pouze jednou a přímo používáme výsledek tohoto podprogramu.

NP-těžký problém Problém Π je NP-těžký (NPH, NP-Heavy), jestliže pro všechny problémy $\Pi' \in \mathrm{NP}$ platí, že $\Pi' \overset{\mathrm{T}}{\propto} \Pi$.

Z definice NPH problémů mimo jiné plyne to , že $\mathrm{NPC} \subset \mathrm{NPH}$.

2.3.3 NPI problémy

NP-intermediate problém Existují problémy, pro které ani neumíme nalézt polynomiální algoritmus, ani je převést na SAT. Takové označujeme jako NPI a patří mezi ně například problém izomorfismu grafů (do roku 2004 taky test prvočíselnosti čísel).

3 Algoritmy

Existuje celá řada neúnosných optimalizačních problémů (NPO), které je ale nutno v praxi řešit. Metody řešení lze rozdělit na *deterministické* a *náhodné a kombinované*. Mezi deterministické patří *aproximativní* (zaručena hodnota chyby v nejhorším případě) a *pseudopolynomiální algoritmy*, metody náhodné a kombinované zastupují *randomizované algoritmy* (s danou statistikou charakteristikou chyby v průměrném případě).

3.1 Pseudopolynomiální algoritmy

Pseudopolynomiální algoritmus Algoritmus, jehož počet kroků závisí polynomiálně na velikosti instance, ale závisí dále na parametru, který s velikostí instance *nesouvisí*.

3.2 Aproximativní algoritmy

Aproximativní algoritmy určují tři třídy aproximovatelných problémů. Třídu aproximovatelných problémů APX, třídu libovolně přesně aproximovatelných problémů PTAS a třídu libovolně přesně a rychle aproximovatelných problémů FPTAS.

3.2.1 Hodnocení kvality algoritmu

Relativní kvalita Algoritmus APR má *relativní kvalitu* R, jestliže

$$R \geq \max_{\forall I} \left\{ \frac{C(APR(I))}{C(OPT(I))}, \frac{C(OPT(I))}{C(APR(I))} \right\}$$

Pro R rovno jedné je algoritmus zaručeně přesný, pro R jdoucí do nekoněčna nedává žádnou záruku přesnosti.

Relativní chyba Algoritmus APR má *relativní chybu* ε, jestliže

$$\varepsilon \geq \max_{\forall I} \left\{ \frac{|C(APR(I)) - C(OPT(I))|}{\max\{C(OPT(I)), C(APR(I))\}} \right\}$$

Pro ε rovno nule je algoritmus zaručeně přesný, pro ε rovno jedné nedává žádnou záruku přesnosti.

Značení ve vzorcích má tento význam:

- $C(S)$ je hodnota optimalizačního kritéria řešení S
- $APR(I)$ je aproximované řešení instance I
- $OPT(I)$ je optimální řešení instance I

Relativní chyba a relativní kvalita jsou vzájemně převoditelné vztahem

$$\varepsilon = 1 - \frac{1}{R}$$

3.2.2 Třídy aproximačních problémů

R-aproximativní algoritmus Algoritmus APR pro problém Π je *R-aproximativní*[1], jestliže je jeho relativní kvalita R.

R-aproximativní optimalizační problém Optimalizační problém Π je *R-aproximativní*, jestliže pro něj existuje R-aproximativní polynomiální algoritmus. Číslo R se nazývá *aproximační práh* problému Π.

APX problém Optimalizační problém Π patří do *třídy problémů APX*, jestliže je R-aproximativní pro konečné R.

PTAS Algoritmus APR, který pro každé $\varepsilon > 0$ vyřeší každou instanci problému Π s relativní chybou nejvýše ε v čase polynomiálním v $|I|$ nazýváme *polynomiální aproximační schéma problému* Π (PTAS, Polynomial Time Approximation Scheme).

PTAS problém Problém Π patří do *třídy PTAS*, jestliže pro Π existuje polynomiální aproximační schéma (patří tam například problém batohu nebo geometrický TSO).

FPTAS Polynomiální aproximační schéma APR, jehož čas výpočtu závisí polynomiálně na $1/\varepsilon$, nazýváme *plně polynomiální aproximační schéma* (FPTAS, Fully Polynomial Time Approximation Scheme).

FPTAS problém Problém Π patří do *třídy FPTAS*, jestliže pro Π existuje plně polynomiální aproximační schéma.

[1] stejným způsobem lze použít relativní chybu ε a mít algoritmus ε-aproximativní

3.2.3 Aproximačně nejtěžší problémy

Zavedením *APX-redukce* tak, že zachovává aproximaci (efektivitu) a chápáním sousloví „efektivní řešení“ jako aproximovatelné řešení lze využít definic z odstavce 2.3 na straně 5 o X-těžkých a X-úplných problémech.

APX redukce APX-redukci značíme jako $\overset{\mathrm{APX}}{\propto}$ a má následující vlastnosti:

- $\Pi_1 \overset{\mathrm{APX}}{\propto} \Pi_2, \Pi_2 \in \mathrm{APX} \Rightarrow \Pi_1 \in \mathrm{APX}$
- $\Pi_1 \overset{\mathrm{APX}}{\propto} \Pi_2, \Pi_2 \in \mathrm{PTAS} \Rightarrow \Pi_1 \in \mathrm{PTAS}$
- APX redukce je Turingova redukce.

NPO-těžký problém Problém Π je *NPO-těžký*, jesliže $\forall \Pi' \in \mathrm{NPO} : \Pi' \overset{\mathrm{APX}}{\propto} \Pi$.

NPO-úplný problém Problém Π je *NPO-úplný*, jestliže je NPO-těžký a $\Pi \in \mathrm{NPO}$.

APX-těžký problém Problém Π je *APX-těžký*, jesliže $\forall \Pi' \in \mathrm{APX} : \Pi' \overset{\mathrm{APX}}{\propto} \Pi$.

APX-úplný problém Problém Π je *APX-úplný*, jestliže je APX-těžký a $\Pi \in \mathrm{APX}$.

3.3 Randomizované algoritmy

Randomizovaný algoritmus je založen na náhodné volbě a jeho vlastnosti jsou vyjádřeny statisticky. Můžeme je rozdělit do dvou skupin:

Monte Carlo algoritmy Dosažený výsledek (optimalizační kritérium) je náhodná proměnná, čas běhu je pro danou instanci pevný.

Las Vegas algoritmy Čas běhu je náhodná proměnná, výsledek je vždy správný.

Výhodou randomizovaných algoritmů je strukturní jednoduchost. Očekávaná kvalita výsledku může být lepší než zaručená kvalita aproximativních algoritmů. Nezávislým opakováním lze kvalitu zlepšit. Tím, že budeme randomizované algoritmy kombinovat s deterministickými prvky, dosáhneme nestrannosti při vzorkování libovolného souboru prvků (každý náhodný start stejně pravděpodobný, každý soused vybrán se stejnou pravděpodobností, ...).

Randomizované paradigma výpočtu ovšem není silnější než deterministické.

4 Lokální metody

Okolí stavu Okolí stavu $s \in S$ je množina stavů, dosažitelných z s aplikací některé operace $q \in Q$.

k-okolí stavu k-okolí stavu $s \in S$ je množina stavů, dosažitelných z s aplikací nejméně jedné a nejvýše k operací $q \in Q$.

Sousední stav Stav $s_0 \in S$ patřící do okolí stavu $s \in S$ se nazývá *sousedním stavem* (sousedem) stavu s.

Lokální metoda má vždy jen jeden aktuální stav, který zpracovává, přičemž uvažuje sousední stavy z (relativně malého) okolí. *Úplné* a *systematické* metody uschovávají každý stav z okolí pro pozdější zpracování. Proto jsou exaktní a proto mohou mít nadpolynomiální složitost.

Konstruktivní heuristika Začne z triviální konfigurace a postupnými kroky konstruujeme řešení.

Iterativní heuristika Začne z nějakého řešení a to postupně vylepšuje.

Dvojfázové heuristiky První fáze slouží k získání řešení (konstruktivně, náhodné řešení), ve truhé fázi iterativní vylepšování.

Jednoduché heuristiky

- Pouze nejlepší
- Prvé zlepšení
- Heuristiky Kernighan-Lin (mají takové to podlouhlé okolí, projdou například pět stavů a pak se podívají, který z těch pěti byl nejlepší a z něj pokračují)

Lokální heuristiky, které neřeší problém lokálních minim, uváznou hned v prvním takovém. To se projevuje tím, že výsledné řešení silně závisí na startovacím stavu. Existuje několik metod, jak uváznutí řešit. Můžeme

- zvětšit okolí,
- startovat z několika různých (náhodných) počátečních řešení,
- vracet se z větví, které nevedou k řešení (detekujeme slepou uličku),
- dočasně připustit zhoršení aktuálního stavu (zvyšuje nároky na řízení algoritmu)

5 Globální metody

Při řešení instancí problému globálními metodami si pomáháme dekompozicí instance na podinstance. Výsledné řešení skládáme z jednotlivých řešení podinstancí. Triviální podinstance řešíme hrubou silou. Přirozeným modelem výpočtu je rekurze.

Základní postup tedy vypadá nějak takto:

$$I \in \Pi \left\{ \begin{array}{lcl} I_1 \in \Pi & \longrightarrow & Y_1 \\ I_2 \in \Pi & \longrightarrow & Y_2 \end{array} \right\} Y$$

instanci problému Π rozložíme na menší instance problému Π a vyřešíme je. Získaná řešení Y_1 a Y_2 pak sloučíme do výsledného řešení Y původní instance problému Π.

Podle specifických vlastností lze dekompozice rozdělit do tří skupin.

Přibližná dekompozice Pokud jsou Y_1 resp. Y_2 optimálními řešeními instancí I_1 resp. I_2, pak je Y řešením (ne nutně optimálním) instance I. Pokud Y_1, Y_2 neexistují, nevíme nic.

Čistá dekompozice Pokud jsou Y_1 resp. Y_2 optimálními řešeními instancí I_1 resp. I_2, pak je Y optimálním řešením instance I. Pokud Y_1, Y_2 neexistují, Y také neexistuje.

Přesná dekompozice Taková čistá dekompozice, že každé optimální řešení Y se dá složit z nějakých optimálních řešení Y_1 a Y_2 (ze všech optimálních řešení Y_1, Y_2 se dají složit všechna optimální řešení Y.

Pokud je navíc dekompozice taková, že jedna z dekomponovaných instancí je triviální, hovoříme o *redukci* (přibližné, čisté, přesné).

5.1 Rozděl a panuj

Algoritmy rozděl a panuj jsou založeny na přibližné dekompozici. Opakované řešení dekomponovaných instancí je řídké. Zvláštním případem jsou metody zmenši a panuj, kdy je potřeba řešit jen jednu z dekomponovaných instancí.

5.2 Dynamické programování

Dynamické programování je čistá dekompozice. Dekomponované instance se dají charakterizovat malým počtem hodnot a řešení dekomponovaných instancí lze těmito hodnotami indexovat. Zároveň se dají dekomponované instance rozdělit do disjunktních tříd (stupňů) tak, že k výpočtu instancí jednoho stupně je třeba jen instancí právě jednoho jiného stupně.

Dynamické programování lze formulovat dvěma způsoby. *Rekurzivní* způsob vyjde ze zadané instance, stanoví, které podinstance je třeba řešit, až dosáhne triviálních instancí. V praxi se tento způsob nepoužívá.

Dopředná formulace také vyjde ze zadané instance, spočítá všechny složitější podinstance, až dosáhne zadané instance.

Řešení jednotlivých podinstancí se samozřejmě v obou případech zaznamenávají a jsou tak vždy počítány nejvýše jednou.

5.3 Složené globální metody

Globální metody lze skládat z různých dekompozic. Principem je rekurzivní procedura, v každé úrovni dekompozice se použije nejlepší možná dekompozice. Výsledkem může být nalezené řešení, informace o tom, že řešení neexistuje, ale také to, že nevíme nic. Tam, kde je možné použít více dekompozic stejného druhu, udržujeme seznam těch, které nevedly k výsledku.

6 Lineární programování

Máme dána reálná čísla $a_{11} \ldots a_{mn}$, $b_1 \ldots b_m$, $c_1 \ldots c_n$. Nalezněte reálná čísla x_1, x_2, $\ldots$, x_n tak, aby

$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

bylo maximální. Zkráceně aby $\boldsymbol{c}^T \boldsymbol{x}$ bylo maximální. Přitom musí být splněna soustava podmínek

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\leq b_2 \\ \vdots &\quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\leq b_m \\ x_1, x_2, \ldots, x_n &\geq 0 \end{aligned}$$

To lze také zkráceně zapsat jako

$$\begin{aligned} \boldsymbol{A} \cdot \boldsymbol{x} &\leq \boldsymbol{b} \\ \boldsymbol{x} &\geq 0 \end{aligned}$$

Do slov přepsáno se jedná o řešení lineárních rovnic, zpravidla s určitým stupněm volnosti, kdy je počet rovnic menší než počet proměnných. Pak se volí některé proměnné jako parametry a dle optimalizačního kritéria se hledá nejlepší řešení (optimum).

6.1 Tvary lineárního programování

Existuje několik tvarů, ve kterých se s lineárním programováním lze setkat.

Kanonický tvar

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} & & \text{maximální} \\ \boldsymbol{A}\cdot\boldsymbol{x} & \leq & \boldsymbol{b} \\ \boldsymbol{x} & \geq & 0 \end{aligned}$$

Standardní tvar

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} & & \text{maximální} \\ \boldsymbol{A}\cdot\boldsymbol{x} & = & \boldsymbol{b} \\ \boldsymbol{x} & \geq & 0 \end{aligned}$$

Obecný tvar

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} & & & \text{maximální} \\ \boldsymbol{A}\cdot\boldsymbol{x} & \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} & \boldsymbol{b} & \\ \boldsymbol{x} & \left\{ \begin{array}{c} \geq \\ \neq \end{array} \right\} & 0 & \end{aligned}$$

Všechny tři tvary jsou vzájemně ekvivalentní. Standardní a kanonický tvar jsou speciálními případy tvaru obecného. Obecný tvar lze převést jak na kanonický, tak na standardní tvar.

Máme-li v původním obecném tvaru dané $\boldsymbol{x} \neq 0$, lze zavést dvě nové proměnné $\boldsymbol{x}^+$ a $\boldsymbol{x}^-$ tak, že $\boldsymbol{x} = \boldsymbol{x}^+ + \boldsymbol{x}^-$ (myšleno po složkách). Pak lze také původní rovnici $\boldsymbol{a}_i \cdot \boldsymbol{x} = b_i$ pro jeden řádek matice $\boldsymbol{A}$ zapsat jako dvě rovnice

$$\begin{aligned} \boldsymbol{a}_i \cdot \boldsymbol{x}^+ & \leq & b_i \\ -\boldsymbol{a}_i \cdot \boldsymbol{x}^- & \leq & -b_i \end{aligned}$$

které již, stejně jako rovnice pro $\boldsymbol{x}^+ \geq 0$ a $\boldsymbol{x}^- \geq 0$, vyhovují kanonickému tvaru.

Podobně lze postupovat i v případě převodu z obecného do standardního tvaru.

6.2 Prostor řešení

Prostorem řešení je konvexní těleso. Aplikací optimalizačního kritéria získáme svazek nadrovnin (rovin), jehož průnik s tělesem prostoru řešení představuje výsledné řešení problému. Průnikem může být

- vrchol (bod),

- hrana (úsečka),
- stěna (nadrovina, rovina).

Pokud nehledáme všechna řešení, stačí zabývat se jen vrcholy. Na množině vrcholů pak provedeme lokální prohledávání (třeba „pouze nejlepší“). Zbývá dané vrcholy nalézt.

Máme-li n proměnných a jen m rovnic, obsahuje matice $\boldsymbol{A}$ nejméně $(n - m)$ lineárně závislých sloupců. Volbou m lineárně nazávislých sloupců volíme *bázi* a všechny proměnné, které neodpovídají sloupcům báze položíme rovny nule. Řešením zbylé soustavy rovnic dostaneme *bázové řešení*.

6.3 Řešíme

Každému vrcholu odpovídá nějaká báze a každé bázi nějaké bázové řešení. Mějme například sedm proměnných $x_1, x_2, \ldots, x_7$ a čtyři rovnice pro omezující podmínky.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= 4 \\ x_1 + x_5 &= 2 \\ x_3 + x_6 &= 3 \\ 3x_2 + x_3 + x_7 &= 6 \end{aligned}$$

Zvolíme-li si potom jako bázi poslední čtyři sloupce matice $\boldsymbol{A}$, můžeme položit $x_1 = x_2 = x_3 = 0$ a dostat tak řešení $\boldsymbol{x} = (0, 0, 0, 4, 2, 3, 6)$.

Tak, teď'ka tedy máme nějaké řešení (odpovídající nějakému vrcholu který odpovídá nějaké bázi). *Cenu* tohoto řešení spočítáme[2] jako

$$z = \sum_{i=1}^{m} x_i c_{Bi}$$

Na další vrchol se posuneme tak, že z báze nějaký sloupec vyhodíme a nahradíme ho jiným. Celé nejlépe tak, že cena výsledného řešení bude vyšší. Postup je docela dobře patrný ze slidů.

Jedna z metod, kterou lze toto dělat, je tzv. *simplexová metoda*. Složitost se odvozuje od počtu možných bází, kterých je $\binom{m}{n}$. Lze ji použít i pro diskrétní problémy, existují ale i metody s polynomiální složitostí.

6.4 Celočíselné lineární programování

U celočíselného lineárního programování jsou všechny proměnné a koeficienty celočíselné (či ve speciálním případě dvouhodnotové 0/1). Řešení celočíselné úlohy má vždy horší optimalizační kritérium než řešení relaxované úlohy v oboru reálných čísel.

7 Slovníček

Hamiltonova kružnice Uzavřená cesta v grafu (posloupnost neopakujících se hran), která obsahuje všechny uzly grafu a žádný v ní není (až na start/cíl) dvakrát.

[2] Proč není v tom vzorečku u ceny index *i* ale nějaký *Bi* mi není moc jasný. IMHO je cena stále stejná, takže nevím, proč by to nemělo jít samotným *i*.